kconfig-webconf: Retrofitting Performance Models onto Kconfig-Based Software Product Lines

Birte Friesel
birte.friesel@uos.de
Universität Osnabrück
Osnabrück, Germany

Kathrin Elmenhorst
kathrin.elmenhorst@rwth-aachen.de
RWTH Aachen University
Aachen, Germany

Lennart Kaiser
lekaiser@uos.de
Universität Osnabrück
Osnabrück, Germany

Michael Müller
michael.mueller@uos.de
Universität Osnabrück
Osnabrück, Germany

Olaf Spinczyk
olaf@uos.de
Universität Osnabrück
Osnabrück, Germany

ABSTRACT

Despite decades of research and clear advantages, performance-aware configuration of real-world software product lines is still an exception rather than the norm. One reason for this may be tooling: configuration software with support for non-functional property models is generally not compatible with the configuration and build process of existing product lines. Specifically, the Kconfig language is popular in open source software projects, but neither language nor configuration frontends support performance models. To address this, we present *kconfig-webconf*: a performance-aware, Kconfig-compatible software product line configuration frontend. It is part of a toolchain that can automatically generate performance models with a minimal amount of changes to a software product line's build process. With such a performance model, *kconfig-webconf* can serve as a performance-aware drop-in replacement for existing Kconfig frontends. We evaluate its usage in five examples, including the busybox multi-call binary and the resKIL agricultural AI product line.

KEYWORDS

Performance Prediction, Regression Trees, Product Lines, Kconfig

ACM Reference Format:
Birte Friesel, Kathrin Elmenhorst, Lennart Kaiser, Michael Müller, and Olaf Spinczyk. 2022. kconfig-webconf: Retrofitting Performance Models onto Kconfig-Based Software Product Lines. In *26th ACM International Systems and Software Product Line Conference - Volume B (SPLC '22), September 12–16, 2022, Graz, Austria.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3503229.3547026

1 INTRODUCTION

Non-functional property (NFP) models are a helpful utility for configuring software product lines (SPLs). When faced with alternative configurations that fulfil the same *functional* requirements, NFP models allow users to predict each configuration's *non-functional* system properties and decide which configuration best suits their needs. For instance, users of resource-constrained embedded devices may prefer small, less efficient algorithms over large libraries optimized for high throughput.

Past research has focused on efficient data acquisition, accurate NFP models, and NFP-aware variability models [5, 9, 11]. Few projects include graphical frontends for NFP-aware configuration; the most prominent example we are aware of is SPL Conqueror [13]. Yet, many practical applications, such as open-source operating systems and other highly configurable product lines, still rely on the Kconfig language and associated frontends for variability modeling and configuration, none of which support NFP models [3]. In fact, the *busybox* project's Kconfig definition goes as far as using in-line annotations of expected binary size instead of a formal NFP model.

In our opinion, this is due to the high cost of adding NFP models and performance-aware system configuration to an existing project. Kconfig is frequently used for header- and Makefile-based feature selection in the open-source world, and different approaches typically require substantial changes to the build system.

Our contribution, *kconfig-webconf*, is meant to minimize these costs. kconfig-webconf serves as an NFP-aware drop-in replacement for existing Kconfig frontends; using it can be as simple as replacing a *kconfig-qconf* call by *kconfig-webconf*. Coupled with a toolchain for automatic generation of NFP models from Kconfig files, this allows for adding NFP models to existing software product lines with a minimal amount of changes to the project.

Among other features, kconfig-webconf can

- show predicted NFP values of the current configuration,
- show how toggling a feature would affect performance, and
- resolve feature dependencies in an NFP-aware manner.

Both kconfig-webconf[1] and our NFP model generation toolchain[2] are available as open-source software under the terms of the GNU AGPL. We provide a short video demonstration online[3].

In the next two sections, we explain our approach for adding NFP models to existing product lines and how *kconfig-webconf* implements it. We present case studies in section 4, and examine related work in section 5 before concluding in section 6.

SPLC '22, September 12–16, 2022, Graz, Austria

ACM ISBN 978-1-4503-9206-8/22/09…$15.00
https://doi.org/10.1145/3503229.3547026

[1] https://ess.cs.uos.de/git/software/kconfig-webconf
[2] https://ess.cs.uos.de/git/software/dfatool
[3] https://ess.cs.uos.de/static/videos/splc22-kconfig-webconf.mp4

2 PERFORMANCE-AWARE FEATURE MODELS

There are two approaches for adding NFPs to a variability model.

Integrated NFP models are part of the variability model: each feature (or group of interacting features) is annotated with its effect on the system's NFPs. The predicted NFP value of a given configuration is the combined effect of the NFP annotation of each active feature or feature group, e.g. the sum of the text segment size of all enabled software components. Examples for this method include ClaferMoo [8] and VELVET [11], and the in-line annotations in the kconfig feature model of the *busybox* open-source project.

External NFP models are black boxes that predict NFPs for complete product line configurations. This requires care to ensure that the NFP model is kept up-to-date after changes in the feature model, but allows for greater flexibility, especially when considering cross-cutting concerns such as optional debug or optimization flags. Examples include L2S [6] and DECART [5].

For kconfig-webconf, we decided on external NFP models. In addition to flexibility and accuracy advantages, these do not require changes in a product line's feature model. In fact, they are independent of the language used for variability modeling, as long as each configuration can be transformed into a formal *feature vector* that can be passed to the NFP model for training and prediction.

Kconfig supports `bool`, `tristate`, scalar (`hex`/`int`), and `string` features. Each feature is defined by at least one `config` block in the Kconfig file. Features whose dependencies are not met are not *visible* and cannot be configured by the user.

We transform a product line instance into a feature vector as follows. Given n distinct features (i.e., `config` entries in the Kconfig file), we define an n-dimensional feature vector $\vec{x}$ and associate each feature with one dimension. We map bool, tristate, and scalar features i to $x_i \in \{0, 1\}$, $x_i \in \{0, 1, 2\}$, and $x_i \in \mathbb{N} \cup \{\perp\}$, respectively. If a boolean or tristate feature is not visible, we set $x_i = 0$, as it cannot be enabled. Scalar features do not have a well-defined value in that case, so we set $x_i = \perp$ instead. For string features, we either use the string literals as-is, or transform them into an enumeration ($x_i \in \mathbb{N}$) for NFP models that only support scalar feature vectors.

As feature vectors can be extracted from the Kconfig and .config files, a repository of .config files and corresponding NFP measurements is sufficient for NFP model generation. The type of NFP model is not relevant for this method – it can be e.g. a classification and regression tree, a simple linear function, or a neural network.

With such an NFP model, all a Kconfig frontend needs to do to become NFP-aware is query the model with the current feature vector whenever the system configuration is changed. Next, we explain how we implement this in kconfig-webconf.

3 USE CASES AND IMPLEMENTATION

The typical product line configuration workflow with Kconfig is as follows. Users use run `make config` or similar to launch a configuration frontend such as `kconfig-nconf` or `kconfig-qconf`. This application reads the feature model from one or more `Kconfig` files, and loads the current configuration from a `.config` file if present.

Users configure the product line, save the new configuration to `.config`, exit the configuration frontend, and run `make` or similar. The build system then uses the `.config` file to select and configure software components and build a product line instance.

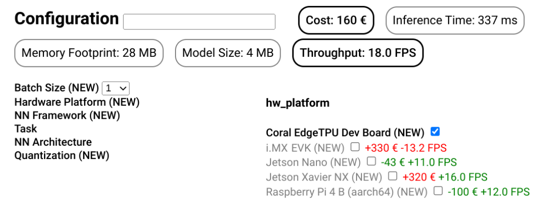

Figure 1: Kconfig-webconf user interface excerpt showing feature model, performance of current configuration (top) and the effect of feature toggles for selected NFPs (right).

In order to make kconfig-webconf usable on a wide range of platforms, we implemented it as a React-based web application. It works entirely client-side, and as such can be used on any platform that is capable of rendering HTML and JavaScript files in a web browser. This includes not just Linux, macOS or Windows computers, but also smartphones and tablets.

In its simplest form, it uses the browser-provided file upload mechanism to load Kconfig, .config and NFP plugin files, and the download mechanism to retrieve the resulting .config file. Alternatively, it can be hosted on a web server that already contains all relevant files, and pass the resulting .config file to an HTTP endpoint instead of offering it for download.

This allows for a wide range of applications. With a few lines of Python, the web server can run locally and access Kconfig and similar files just like any CLI or desktop application would, thus serving as a true drop-in replacement for `make config`.

With a more involved setup, kconfig-webconf can run on a web server, pass user-configured .config files to a build service, and offer the resulting product (e.g. application binary) for download instead. This allows users to configure product line instances without needing to worry about build system details and build dependencies, and without access to source code.

Fig. 1 shows an excerpt of the kconfig-webconf GUI when configuring an AI product line [4]. In addition to the regular Kconfig menu, it shows the calculated NFP values of the current configuration. In this excerpt, the user has selected *cost* and *throughput*, causing kconfig-webconf to annotate each feature toggle with its effect on these two non-functional properties. Here, we see that the Jetson Nano platform is both cheaper and faster than the currently selected EdgeTPU board, so switching to it may be a good idea.

We now explain how we implemented feature model and NFP support in kconfig-webconf.

3.1 Feature Model

In order to avoid issues with ill-defined Kconfig semantics [3], we consider the Linux Kconfig parser and frontends as reference implementations and aim to imitate their behaviour.

We express each feature as a *KNode* object and augment it with dependencies, (conditional) defaults and reverse dependencies, and attributes inherited from parent nodes. Additionally, each KNode holds its current value, and a flag indicating whether it has been user-configured or set due to a `default` or `select` statement. Based

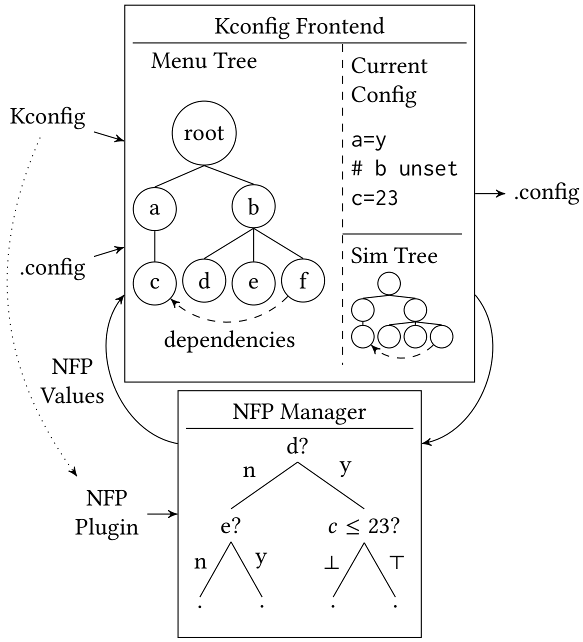

Figure 2: Data flow within *kconfig-webconf*: input (left), internal data structures (middle), and output (right).

on both explicit (`choice`, `menu`, and `menuconfig` blocks) and implicit (`depends on`) annotations, we arrange the KNodes in a tree structure representing the configuration tree that is visible to users.

We do not translate the Kconfig specification into logic formulas or extend it otherwise: when used without NFP models, kconfig-webconf is meant to behave like any other Kconfig frontend.

3.2 NFP Models

We implement NFP models as *plugins* that interface with Kconfig via the feature vectors mentioned in Section 2. An NFP plugin is a JSON file that describes the Kconfig file it belongs to by means of a checksum and feature names, and that holds at least one NFP model. Each model predicts a single non-functional property and indicates whether minimizing or maximizing that property is desirable. Fig. 2 shows how NFP plugins and Kconfig feature model interact.

kconfig-webconf's *NFP Model* class contains code for evaluation of NFP models (i.e., turning feature vectors into predicted NFP values). It is the only file that needs to be changed in order to support additional types of NFP models. Currently, we support feature annotations, classification and regression trees (CART), and regression model trees (RMT); however, kconfig-webconf can easily be extended for additional NFP model types.

The *NFP Manager* is responsible for interfacing between GUI, feature model (Kconfig tree), and NFP models. It holds references to all NFP models, the Kconfig menu tree (i.e., the current configuration), and a tree copy used for simulating configuration changes.

Each user interaction in the Kconfig tree (loading a .config file or changing a feature) triggers the NFP Manager, which in turn passes the Kconfig menu tree to each NFP model for evaluation, and displays the estimated NFP value in the user interface (Fig. 1, top). Additionally, for each visible boolean feature, it simulates the effect of a user toggling that feature using its internal tree copy, and calculates the NFP value for the simulated configuration. This allows kconfig-webconf to show the relative effect of toggling each visible feature on each non-functional property, and indicate whether toggling the feature is desirable from an NFP optimization point of view (see the +/- annotations in Fig. 1).

3.3 Limitations

kconfig-webconf is a by-product of our research on exploiting NFP knowledge during product line configuration. As such, the focus is on evaluating NFP model types and NFP-aware configuration strategies, and we cannot guarantee full Kconfig compatibility yet.

Notably, as it is a web application, kconfig-webconf does not support `source` statements that are used to include additional Kconfig files. We provide a pre-processor script that resolves source statements and outputs a single, otherwise identical Kconfig file instead. We also do not support the Kconfig macros used by the Linux kernel, as these are rarely used in other product lines. Nevertheless, we expect that these can also be implemented in a preprocessor.

Even with these limitations, we consider kconfig-webconf to be a helpful utility; its open-source nature allows anyone to use and improve it. We now show cases where we have used it in the past.

4 CASE STUDIES

We originally developed kconfig-webconf for the *resKIL* agricultural AI product line [4]. It is part of a project that uses computer vision systems for classification and semantic segmentation on resource-constrained embedded devices. We are, for example, interested in sweet spots of low hardware cost, high inference throughput, and acceptable model accuracy.

Our feature model describes the hardware platforms, AI platforms and quantization settings, AI architectures, and runtime settings (batch size) that can be used to perform classification or segmentation tasks. The NFP models extend it with hardware cost, model accuracy, inference latency and throughput, memory usage, and model size. With kconfig-webconf, we can show stakeholders how the hardware platforms they are currently considering affect system performance vs. hardware cost, and allow them to decide on promising AI architectures for transfer learning.

Additionally, an experimental Pareto optimization feature allows us to configure relevant functional requirements (e.g. semantic segmentation or a certain AI architecture) and automatically determine the set of optimal implementations.

A kconfig-webconf demo site using the resKIL feature model and NFP models is available online[4]. Fig. 1 shows an excerpt.

To verify that kconfig-webconf can easily be used with existing product lines, we generated ROM and RAM usage models for the open-source *busybox* multi-call binary, our in-house *Kratos* and *Multipass* operating systems, and the *MxKernel* project [7].

In all cases, we only needed to add three targets to the existing project Makefiles: `make randconfig` for random sampling of the configuration space, which is sufficient for regression-tree based NFP model generation [5]; `make nfpkeys` to describe which non-functional properties are being evaluated; and `make nfpvalues` to

[4]https://ess.cs.uos.de/git-build/kconfig-webconf/master/reskil-rmt/

measure and output the NFP values of the current configuration. In fact, busybox already provides a suitable randconfig target.

The resulting models allow for NFP-aware configuration of these projects. In case of busybox, which already has in-line annotations indicating how individual features affect the binary size, the kconfig-webconf model is more accurate and more detailed: in contrast to our regression tree-based models, the in-line annotations are by design incapable of expressing feature interactions of cross-cutting concerns such as debug flags [12]. Additionally, busybox only annotates multi-call binary components such as syslogd, but does not annotate configuration options of those (e.g. log rotation or remote log support).

Overall, we found NFP model generation and replacing existing Kconfig frontends with kconfig-webconf to be a low-effort task with immediately noticeable advantages.

5 RELATED WORK

Although there has been a considerable amount of research covering sampling strategies for NFP model generation and NFP modeling methods, NFP-aware variability modeling languages appear to be rare, and NFP-aware SPL configuration software even more so [10]. As such, we consider kconfig-webconf's ability to use NFP models that are independent of the feature modeling language to be a useful addition to the state of the art.

SPL Conqueror is a prime example of NFP-aware SPL tooling [13]. It encompasses the whole life cycle from SPL and NFP definition over benchmarking to NFP-aware configuration and optimization. This makes it more powerful than kconfig-webconf, but also more complex, as it uses a custom variability modeling language. Considering this, it may in fact be too powerful for wide-spread adoption.

Additional examples are *ClaferMoo* [8] and *TVL* [2]. Both rely on per-feature NFP annotations that must be provided by domain experts rather than automated benchmarks. Although these are more expressive than the ones used in busybox (and part of the language syntax rather than in-line annotations in a language without NFP support), their flexibility is limited compared to kconfig-webconf's support for nearly arbitrary NFP model types.

The *VM* language also has detailed support for feature attributes such as non-functional properties [1]. However, it focuses on automatic generation of diverse product line instances for NFP measurements rather than NFP-aware configuration, and does not include a configuration frontend.

6 CONCLUSION

We have presented *kconfig-webconf*, an NFP-aware configuration frontend for Kconfig-based software product lines. In contrast to other approaches, kconfig-webconf explicitly distinguishes between an NFP-unaware variability model and an external NFP model, and combines them into an NFP-aware configuration frontend. This gives it high flexibility, as the type of NFP model can be easily adjusted towards domain-specific requirements.

In a case study of four software product lines and one AI product line, we have shown that adding kconfig-webconf to existing software projects is typically as easy as implementing three additional `make` targets for benchmarking, and replacing the call to an existing Kconfig frontend with a kconfig-webconf wrapper. We found that using it provides valuable insights to developers and stakeholders. kconfig-webconf and our Kconfig-based NFP model generation toolchain are available as open-source software.

ACKNOWLEDGMENTS

This paper was supported by funds of the Federal Ministry of Food and Agriculture (BMEL) based on a decision of the Parliament of the Federal Republic of Germany via the Federal Office for Agriculture and Food (BLE) under the innovation support programme.

REFERENCES

[1] Mauricio Alférez, Mathieu Acher, José A Galindo, Benoit Baudry, and David Benavides. 2019. Modeling variability in the video domain: language and experience report. *Software Quality Journal* 27, 1 (2019), 307–347. https://doi.org/10.1007/s11219-017-9400-8

[2] Quentin Boucher, Andreas Classen, Paul Faber, and Patrick Heymans. 2010. Introducing TVL, a text-based feature modelling language. In *Proceedings of the Fourth International Workshop on Variability Modelling of Software-intensive Systems (VaMoS'10), Linz, Austria, January*. 159–162.

[3] Sascha El-Sharkawy, Adam Krafczyk, and Klaus Schmid. 2015. Analysing the Kconfig Semantics and Its Analysis Tools. In *Proceedings of the 2015 ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences* (Pittsburgh, PA, USA) *(GPCE 2015)*. Association for Computing Machinery, New York, NY, USA, 45–54. https://doi.org/10.1145/2814204.2814222

[4] Birte Friesel and Olaf Spinczyk. 2022. Black-Box Models for Non-Functional Properties of AI Software Systems. In *Proceedings of the 1st Conference on AI Engineering - Software Engineering for AI (CAIN'22)* (Pittsburgh, PA, USA) *(CAIN'22)*. 11 pages. https://doi.org/10.1145/3522664.3528602 to appear.

[5] Jianmei Guo, Dingyu Yang, Norbert Siegmund, Sven Apel, Atrisha Sarkar, Pavel Valov, Krzysztof Czarnecki, Andrzej Wasowski, and Huiqun Yu. 2018. Data-Efficient Performance Learning for Configurable Systems. *Empirical Softw. Engg.* 23, 3 (June 2018), 1826–1867. https://doi.org/10.1007/s10664-017-9573-6

[6] Pooyan Jamshidi, Miguel Velez, Christian Kästner, and Norbert Siegmund. 2018. Learning to Sample: Exploiting Similarities across Environments to Learn Performance Models for Configurable Systems. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Lake Buena Vista, FL, USA) *(ESEC/FSE 2018)*. Association for Computing Machinery, New York, NY, USA, 71–82. https://doi.org/10.1145/3236024.3236074

[7] Michael Müller and Olaf Spinczyk. 2019. MxKernel: Rethinking Operating System Architecture for Many-core Hardware. In *9th Workshop on Systems for Multi-core and Heterogenous Architectures*. Dresden, Germany.

[8] Rafael Olaechea, Steven Stewart, Krzysztof Czarnecki, and Derek Rayside. 2012. Modelling and Multi-Objective Optimization of Quality Attributes in Variability-Rich Software. In *Proceedings of the Fourth International Workshop on Nonfunctional System Properties in Domain Specific Modeling Languages* (Innsbruck, Austria) *(NFPinDSML '12)*. Association for Computing Machinery, New York, NY, USA, Article 2, 6 pages. https://doi.org/10.1145/2420942.2420944

[9] Juliana Alves Pereira, Mathieu Acher, Hugo Martin, Jean-Marc Jézéquel, Goetz Botterweck, and Anthony Ventresque. 2021. Learning software configuration spaces: A systematic literature review. *Journal of Systems and Software* 182 (2021), 111044. https://doi.org/10.1016/j.jss.2021.111044

[10] Rick Rabiser, Klaus Schmid, Martin Becker, Goetz Botterweck, Matthias Galster, Iris Groher, and Danny Weyns. 2018. A Study and Comparison of Industrial vs. Academic Software Product Line Research Published at SPLC. In *Proceedings of the 22nd International Systems and Software Product Line Conference - Volume 1* (Gothenburg, Sweden) *(SPLC '18)*. Association for Computing Machinery, New York, NY, USA, 14–24. https://doi.org/10.1145/3233027.3233028

[11] Marko Rosenmüller, Norbert Siegmund, Thomas Thüm, and Gunter Saake. 2011. Multi-Dimensional Variability Modeling. In *Proceedings of the 5th Workshop on Variability Modeling of Software-Intensive Systems* (Namur, Belgium) *(VaMoS '11)*. Association for Computing Machinery, New York, NY, USA, 11–20. https://doi.org/10.1145/1944892.1944894

[12] Norbert Siegmund, Sergiy S. Kolesnikov, Christian Kästner, Sven Apel, Don Batory, Marko Rosenmüller, and Gunter Saake. 2012. Predicting performance via automated feature-interaction detection. In *2012 34th International Conference on Software Engineering (ICSE)*. 167–177. https://doi.org/10.1109/ICSE.2012.6227196

[13] Norbert Siegmund, Marko Rosenmüller, Martin Kuhlemann, Christian Kästner, Sven Apel, and Gunter Saake. 2012. SPL Conqueror: Toward optimization of non-functional properties in software product lines. *Software Quality Journal* 20, 3-4 (2012), 487–517. https://doi.org/10.1007/s11219-011-9152-9